Evolutionary Methods in Self-organizing System Design

István Fehérvári and Wilfried Elmenreich
Institute for Networked and Embedded Systems, University of Klagenfurt, Klagenfurt, Austria

Abstract— *Self-organizing systems could serve as a solution for many technical problems where properties like robustness, scalability, and adaptability are required. However, despite all these advantages and due to the decentralized control there is no straight-forward way to design such a system. In this paper we describe a novel design approach using genetic algorithms and artificial neural networks to automatize the part of the design process that requires most of the time. A simulated robot soccer game was implemented to test and evaluate the proposed method. A new approach in evolving competitive behavior is also introduced using Swiss System instead of the full tournament to cut down the number of necessary simulations.*

Keywords: Self-organizing Systems, Genetic algorithm, Artificial neural networks, Swiss System tournament

1. Introduction

Following the continuous technical development in almost any areas, systems are getting more and more complicated and as a result the need for adaptability, robustness and scalability is increasing. An alternative solution could be the usage of self-organizing systems (SOS) where these properties emerge from simple interactions between the system's components. Typically, the emergent service is really hard, or even impossible to predict. Thus, finding a set of rules that causes the overall system to exhibit the desired properties presents a great challenge to the system designers. The main problem is that a small change of a parameter might even lead to counter-intuitive results. In [1], M. Resnick presents an example of a simple simulation of slime mold behavior and was asking colleagues to predict the effect of a simple change in the rule set. There were only two possible answers, so without knowledge, there was a 50% chance to guess the right answer. However, in the experiment a significant majority of answers, including those from experts on SOS behavior, was wrong.

To design a self-organizing system with the desired emergent behavior, it is crucial to find local rules for the behavior of the system's components (agents) that generate the intended behavior at system scale. In many cases, this is done by a sumptuous trial and error process which in case of systems with high complexity is not efficient or even unfeasible. Parameter intensive systems also suffer from the problem of the unpredictable result of one small change in the parameter set. Novel approaches like evolutionary methods provide means to automatize the testing and optimizing of parameters in an intelligent way.

The objective of this paper is to propose a new method for designing self-organizing systems using evolutionary computing, especially genetic algorithms (GA) as a parameter search technique. A robot soccer simulation model has been created, where the local rules -agent behaviors- are represented in an evolvable fashion by using artifical neural networks (ANN) that process the sensory inputs and generates the control outputs of a player. The aim is to use genetic algorithms on the ANNs to automatize the process of designing the control system and, thus, decreasing the necessary human interaction.

The paper is structured as follows: In the next section an overview about self-organization and systems presenting the background of this paper will be given. Then the design steps as a general approach with the genetic algorithms will be described. Section 4 focuses on the practical evaluation of the approach, presenting the setup of the robot soccer simulation while Section 5 shows and explains the acquired results. Related work is discussed in Section 6. This paper is concluded in Section 7.

2. Self-organizing Systems

The concept of SOS was first introduced by W. Ross Ashby in 1947 referring to pattern-formation processes taking place within the system by the cooperative behavior of its individuals. These could be best described by the way they achieve their order and structure without having any external directing influences. There are several definitions for SOS[2], the following was formulated on the Lakeside Research Days'08:
"*A self-organizing system (SOS) is a set of entities that obtains global system behavior via local interactions without centralized control.*"
An example could be a team of workers acting on their own following a mutual understanding. If there were any external influence like a common blueprint or a boss giving orders it would result in no self-organization. Also many examples of SOS can be observed in nature. A school of fish where each individual has knowledge only about its neighbors and so there is no leader amongst them. The key part is the communication between the individuals; the way they satisfy their own goal such as getting close, but not too close to other fish in the school while trying to find food in the water.

Furthermore it is important to note that the emergent property cannot be understood just by simply examining the system's components in isolation, but requires the consideration of the interactions among the components. This interaction is not necessity direct, it can be indirect as well when one component modifies the environment which then influences other components. This presents a continuous mixture of positive and negative feedback in the behavior. In the mentioned example positive feedback is the attractive effect of the entities in the school or a visible trace of food; the negative feedback is the proximity of other fish.

By describing self-organizing systems the following advantages can be observed: These systems are often very robust against external disturbances; it is clear that a failure of one component rarely results in a full collapse. Also adaptability and scalability can be noticed meaning a dynamical behavior and a flexibility in the number of components. These properties make SOS an interesting option for controlling complex systems, however these and the decentralized control makes an SOS difficult to design and control. Although some ideas for design exist, there is no general methodology yet explaining how these should be done.

3. General Methodology

This section describes the proposed method trying to give a tool for SOS design engineers. The process starts by defining the main goals: what are the expectations from our system. The next step is to build an evolvable representation of local behavior. Our approach uses a genetic algorithm to explore the solution space, therefore the evolvability of the representation is of outmost importance so GA operators like mutation and/or crossover must be defined on it. Usually, control software is written in programming languages (JAVA, C, etc.) which are inappropriate for direct mutation and crossover operators. One notable exception could be the LISP programming language which is used for the representation of an evolvable algorithm in [3]. Unlike standard programs, artificial neural networks (ANNs) or representations based on fuzzy rules are qualified for this task. Structure and setup of this representation is also a question; it can be also trained by the genetic algorithm or defined in advance (see Figure 1).

In case of ANNs reinforced learning is needed, since we have to deal with belated rewards we get after a simulation of many steps of the revised ANN. Thus, the standard backpropagation algorithm cannot be applied to program the ANN's weights. At this point we need our goals to be formulated as rewards for reinforcing the candidates. We propose a step by step approach decomposing the overall goal into smaller achievements weighted according to their significance to make the learning process smoother. A typical example would be an object manipulation task for a robot where the three subtasks would be: finding, grabbing and then manipulating the object. In the next chapter an example of this methodology will be given.

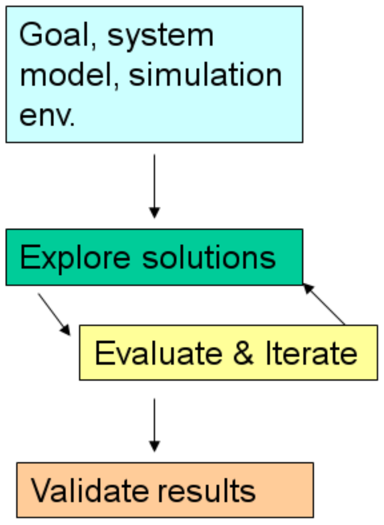

Fig. 1: Design method

With a simulation environment acting as a playground, the genetic algorithm can start evolving the possible candidates. Typically, a fitness value can be deduced from the simulation results. This fitness value is then used in the GA to decide on the fittest individuals. An example for such a fitness value could be the throughput of a given setup of a wireless network. In many cases an absolute fitness value cannot be assigned especially when a competitive emergent behavior is expected. In order to get a ranking on the individuals it is necessary to play a tournament among the candidates of a population. In a native approach, the number of pairings equals $n(n-1)/2$; n being the number of individuals in a population. In case of long simulations the time can be effectively cut by using the swiss system, a pairing system used to organize (chess) tournaments which yield a ranking with a minimum number of pairings[4] instead of full tournament. Detailed description can be found in the next chapter.

If the fitness definition was correct the achievements should emerge one after the other otherwise a reconsideration of the fitness function is necessary.

4. Evolving Simulated Soccer Agents

As a practical example simulated robot soccer has been chosen because it reflects complex control problems and by regarding all the players equal (without dedicated roles such as, for example, a goalkeeper) it gives a very suitable example for self-organizing systems. The first idea was to use the official Robot Soccer Simulator in the training process, however since it runs in real-time the time needed for a sufficient evolution was unacceptably high. To overcome this

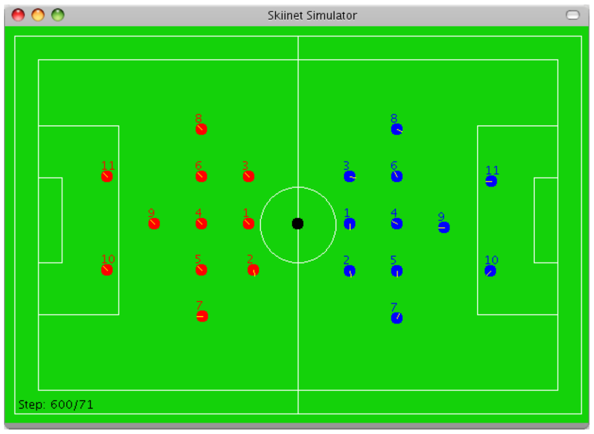

Fig. 2: Robot soccer simulator

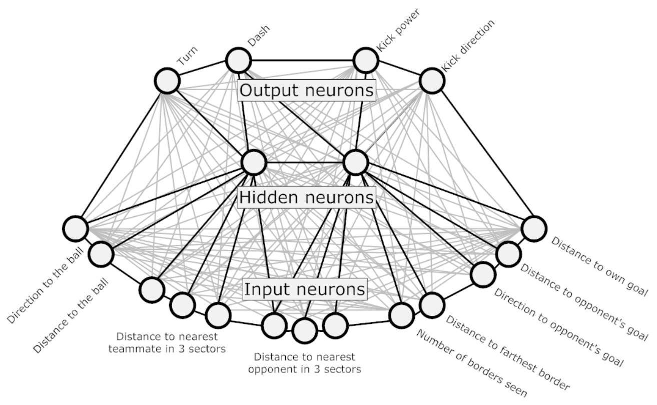
Turn
Dash
Kick power
Kick direction
Output neurons
Hidden neurons
Input neurons
Direction to the ball
Distance to the ball
Distance to nearest teammate in 3 sectors
Distance to nearest opponent in 3 sectors
Number of borders seen
Distance to farthest border
Direction to opponent's goal
Distance to opponent's goal
Distance to own goal

Fig. 3: A possible layout of a fully connected network with two hidden neurons

problem an own simulation environment has been created which generates the same input for the same output of the teams as played in the offical server but using asynchronic coupling not to lose time between cycles.

A game is played by two teams of eleven players each for 60 seconds trying to score goals. Players have to respond to their received visual information every cycle which corresponds to 200ms in real-time according to our definition. We also made the simulation simpler compared to the official one by removing the rules for kick off, corner kick, side kick and penalty kick from the game. Thus, the soccer players have the ability to turn a given relative angle, to dash to the direction where they are heading and to kick the ball with a given power and direction. It was also necessary to program the agents to kick the ball when it is in proximity meaning they will always kick the ball when they can. In case the ball gets out of the field it will be immediately placed back to the center. The described changes do not affect the validation of the generality of our approach. Also, our goal was not to generate a competitive robot soccer team, but to find a suitable and realistic testbed for our proposed approach. Figure 2 shows the simulator display output.

4.1 Neural Network Controller

The control of the agents was realized by a fully connected, time-discrete, recurrent artificial neural network. Each neuron is connected to every other neuron and itself via several input connectors. Each connection is assigned a weight that is a floating value and each neuron is assigned a bias. We used 13 inputs, 4 outputs (see Table 1) and different sets of hidden neurons (4-6-8). The inputs are updated every cycle and two steps of the network were executed. The second step is required for the update of the hidden neurons. At each step, each neuron i builds the sum over its bias b_i and its incoming weights w_ji multiplied by the current outputs of the neurons $j = 1, 2, ..., n$ feeding the connections. The output of the neuron for step $k + 1$ is calculated by applying an activation function F:

$$o_i(k+1) = F(\sum_{j=0}^{n} w_{ji} o_j(k) + b_i)$$

where F is a threshold function.

$$F(x) = \begin{cases} 180 & \text{if } x \geq 180 \\ x & \text{if } -180 < x < 180. \\ -180 & \text{if } x \leq -180 \end{cases}$$

The thresholds of -180 and 180, respectively, have been chosen in order to make a conversion to a +/- 180 degree turning angle suitable for kick direction and turning. For dash and kick power the simulator implements a threshold of 100, thus, values above this value are truncated. The neural network learns to adjust its outputs accordingly, therefore, the output of the respective neurons can be fed directly into the simulator without the need for explicit conversion. It is important to note that the players have no global positioning information about themselves, any other players or the ball. Their inputs are depending on relative information gained by their simulated visual sensor. Players have a visual range of 90 degrees split up to three sectors: from -45 to -15 (Sector 1), from -15 to 15 (Sector 2) and from 15 to 45 (Sector 3) degrees. These sectors help the player coordinate its movement on the field. There are also two inputs (9-10) responsible for sensing the edge of the field. The first one indicates the number of borders seen by the player which is the number of crossections defined by the line of sight and the border of the field. It can take the following values:

- 0: player is out of the field, looking away
- 1: player is in the field
- 2: player is out of the field, looking to the field

Training has been conducted by using a genetic algorithm on the weights and biases of the network with the parameters described by Elmenreich and Klingler[5]. Basically, multiple solutions are created using stochastic methods and evaluated while the best ones are selected for the next generation. We built on the versatile framework developed by Pfandler [6] that supports mutation, crossover, elite selection, random selection, and co-evolution of multiple populations. Each version of an ANN is represented by the weight matrix and the biases of each neuron, which we also call the *genome* of the network.

In our setup, the selection applies elite selection by keeping the top 15% of networks for the next generation. Furthermore, a random selection selects another 12%, where networks with higher scores and greater diversity for the gene pool have a higher chance of being selected. 20% of the population are mutated, 48% are filled with offspring from crossover functions on the already selected networks, and 5% of the population are replaced by randomly created new entities. We experimented with population sizes between 25 and 100. The multiple population feature was not used.

Neurons	Inputs	Outputs
1	Distance to the ball	Dash
2	Relative direction of the ball	Turn
3	Dist. to nearest teammate in sector #1	Kick power
4	Dist. to nearest teammate in sector #2	Kick direction
5	Dist. to nearest teammate in sector #3	
6	Dist. to nearest opponent in sector #1	
7	Dist. to nearest opponent in sector #2	
8	Dist. to nearest opponent in sector #3	
9	Number of borders seen	
10	Distance to farthest border	
11	Distance to own goal	
12	Distance to opponent's goal	
13	Relative direction to opponent's goal	

Table 1: Neural network setup

4.2 Step by Step Fitness

Typically when the task is more complex the definition of the fitness function is not trivial. In the case of robot soccer the primary aim is to train teams scoring the most goals during the given time. However the problem is far too complicated for a team to find the best solution just by rewarding by the number of scores. The idea is to decompose the overall goal into smaller achievements (so-called guidelines) and let the teams fulfill them one after the other. This method tries to ensure a smooth learning process assuming some preliminary knowledge or ideas about the solution. The guidelines are assigned a weight to setup a hierarchical order among them. It means a task with smaller weight is less important, but will be most likely be accomplished before another tasks with higher value. This is because the second one is too complex to be achieved without the first one. Figure 4.2 shows the applied tasks in their respective order in our simulation.

At the beginning of the training we wanted the teams to learn that a good distribution on the field might lead

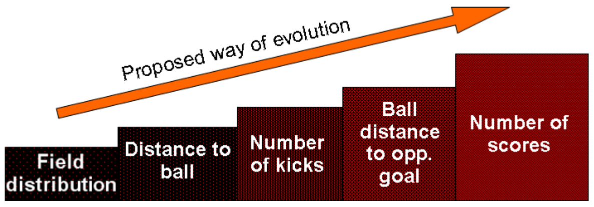

Fig. 4: Weighted fitness

to good overall play. This lead to the introduction of the first guideline. It was implemented by defining 64 evenly distributed checkpoints on the field and counting the number of controlled points every 5 seconds for both teams. A point is controlled by a team if it has the nearest player to this point. The accumulated points were added to the final fitness value. The second guideline was an advice for the teams to move their players closer to the ball. The distance of the nearest player for both teams to the ball is measured and compared every 4 seconds. The team having a player closer to the ball earned one point. At the end of the game this point was weighted and also added to the final fitness. The number of kicks was also counted with a weight however kicking out of field gives only half the points. It must be also noted that only the first ten kicks were rewarded to avoid a local minima where one player is constantly kicking the ball out while the others occupying the checkpoints. Concerning the kicking direction the ball distance to the opponents goal was also measured and calculated every 2 seconds in the same manner as guideline two. The highest weight was assigned to the number of scores making insignificant all the actions by the other team without a score.

4.3 Tournament Ranking with Swiss System

Evolving competitive team behavior is a good example where one cannot assign a simple absolute fitness value. To get a ranking on the teams a solution is to play a tournament among the candidates in each generation (assuming one population). A full tournament would mean $n(n-1)/2$ number of pairings when n is the number of entities in the population. In case a simulation run takes too much time or a high number of generations is needed this approach can be very ineffective. For example, a population of 50 individuals would require 1225 runs each generation. The proposed solution tries to minimize the number of necessary pairing using Swiss System style tournament[7]. It reduces the required number to $\lceil \log_2 n \rceil \frac{n}{2}$ which is in the mentioned case only 150 games (see figure 5). Inspired by the official FIDE[1] rules for chess tournaments we established the following system:

In each game the winner gets two points, loser gets zero, in case of a draw both get one point. After the first round players are placed in groups according to their score

[1]http://www.fide.com

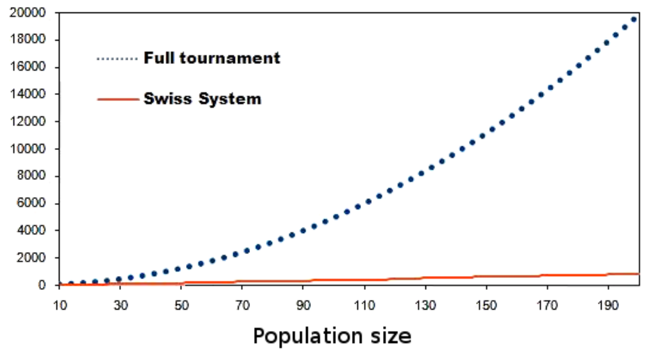

Fig. 5: Total number of games in full tournament and Swiss System

(winners in the 2 group, those who drew go in the 1 group, and losers go in the 0 group). The aim is to ensure that players with the same score should play against each other. Since the number of perfect scores is cut in half each round, it doesn't take long until there is only one player remaining with a perfect score. The actual number of rounds needed is n to be able to handle 2^n number of players. In chess tournaments there are usually many draws, so more players can be handled (a 5 round event can usually determine a clear winner for a section of at least 40 players, possibly more), although in our simulation a draw is very unlikely. To avoid early games between elite selected entities the first round is not randomized but cut into two halves where the first half is playing against the second half.

The drawback of the Swiss system is that it is only designed to determine a clear winner in just a few rounds. Regarding other players, we have little information about their correct ranking. For example, there could be many players with 3-2 scores and it is hard to say which player is better than the others, or whether a player with 3.5 points is better than a player with 3 points. To help determine the order of finish, a tiebreak method has been implemented. In order to decide on the ranking for players having same scores, we used a method developed by Bruno Buchholz[8]. There the score of the players' opponents is summed up thus favoring those who have confronted better opponents. In case it is still undecided the sum is extended by the points of those opponents who have lost against the player. This uncertainty in the ranking could cause problems in GA operators when selecting entities for survival to the next generation. In our case elite selection was 15% while the Swiss System ensures only the first and last position to be ranked correctly (compared to a full tournament of the same population), thus the position of all other players carries also some obscurity. After observing this effect in our simulation we came to the conclusion that having a somewhat imprecise selection among the top players slows down the process just a little or not at all. To select entities for survival we used a roulette wheel selection where the probability being selected is directly proportional to the fitness, in our case the ranking of the Swiss System. Since this approach already carries some randomization some more uncertainty did not make a crucial impact.

5. Experiences

As described in Section 4 a robot soccer simulation model has been created where the aim was not to evolve highly competitive robot soccer teams but to provide a suitable testbed, therefore the evolved behavior will not be analyzed specifically in this context. Two experimental setups with different population numbers have been used to show their evolution progress during approximately the same time (same number of total games) as seen in Table 2. In our scenario where there is no absolute fitness value assigned to the players due to the tournament ranking method it is hard to track the progress of the evolution. This is important to recognize when a local or global minima has been reached making further computations pointless. An appropriate solution was to save the best entity in every 10th or 100th generation and simply making them play against each other in a full tournament manner where they were ranked according to the number of games won.

We experienced that a continuous evolution was sustainable till it reached a local cost minimum. It means to reach a certain level of advancement especially for more complex systems having more waypoints in their way of evolution the number of iterations is a crucial factor. By using Swiss System instead of full tournaments the number of simulation runs were greatly reduced. In the first setup the acceleration was 74% while in the second one with a population of 50 it was 87% saving a considerable amount of time taken just for simulation runs.

No.	Pop. size	Generations	Total number of games	Acceleration rate
1	20	5000	250 000	74%
2	50	2000	300 000	87%

Table 2: Simulation scenarios

6. Related Work

There have been some proposals for designing self-organizing systems. First a method proposed by Gershenson[9] introducing a notion of "friction" between two components as a utility to design the overall system using trial and error. Methods building on trials even if they are improved by certain notions often suffer from counter-intuitive interrelationships between local rules and emergent behavior. There is an imitation-based approach proposed by Auer[10] where the behavior of a hypothetical omniscient "perfect" agent is analyzed and mimicked in order to derive the local rules. The problem here and in all methods based on imitation is the limitation to cases where an appropriate example model is available.

So far there exist almost no related work aiming specifically on the evolution of cooperative systems or self-organizing systems based on ANN controllers. A notable exception is the work of Nelson[11], describing the evolution of multiple robot controllers, also using full tournament rankings, towards a team that plays “Capture the flag“ against an opponent team of robots. Baldassarre[12] also shows interesting results in evolving physically connected robots using ANN controllers but no tournament rankings.

7. Conclusion and Future Work

In this paper a new method for designing self-organizing systems was described and tested which is based on genetic algorithms and a special ranking system. As a testbed a simulated robot soccer game was implemented where the players were controlled by artificial neural networks bred by a genetic algorithm. For the learning procedure reinforced learning was used where the main goal was divided into several sub-achievements acting as waypoints or guidelines for the entities to find a suitable solution for the given problem. Fitness was not directly measured but a tournament was created in each population to rank all the entities. As seen in [11], full tournament is applicable when no direct fitness can be calculated however for more complex systems the number of iterations could be really high resulting to a long simulation time especially with high population sizes. The main contribution of this paper is the usage of a Swiss System in the tournament phase which greatly shortens the simulation time by reducing the number of pairings but not hindering the overall process. This work will be continued by a more detailed analysis on the Swiss System and its application in genetic algorithms, especially in evolving competitive behavior. It is also an interesting research question how to find the best and minimal number of guidelines making the design process more generic. Novel methods on finding better GA settings in this topic will be also investigated.

Acknowledgment

This work was supported by the Austrian FWF project TTCAR under contract No. P18060-N04. We would like to thank our colleague Johannes Klinglmayr for contructive comments on an earlier version of the paper.

References

[1] M. Resnick, *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds (Complex Adaptive Systems).* The MIT Press, January 1997.

[2] S. Camazine, J. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau, *Self-Organization in Biological Systems*, 2nd ed., P. W. Anderson, J. M. Epstein, D. K. Foley, S. A. Levin, and M. A. Nowak, Eds. Princeton, NJ, USA: Princeton University Press, Jan. 2001, vol. 1.

[3] J. R. Koza, Ed., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, 1st ed. The MIT Press, Dec. 1992.

[4] T. Just and D. B. Burg, Eds., *U.S. Chess Federation’s official rules of chess*, 5th ed. New York: Random House Puzzles & Games, 2003.

[5] W. Elmenreich and G. Klingler, “Genetic evolution of a neural network for the autonomous control of a four-wheeled robot,” in *Sixth Mexican International Conference on Artificial Intelligence (MICAI’07)*, Aguascalientes, Mexico, Nov. 2007.

[6] A. Pfandler, “Design and implementation of a generic framework for genetic optimization of neural networks,” Bachelor’s Thesis, Vienna University of Technology, Vienna, Austria, 2007.

[7] “FIDE swiss rules,” Approved by the General Assembly of 1987. Amended by the 1988 & 1989 General Assemblies.

[8] Wikipedia, “Buchholz system — Wikipedia, the free encyclopedia,” [Accessed 22-Mar-2009]. [Online]. Available: http://en.wikipedia.org/wiki/Buchholz_system

[9] C. Gershenson, “Design and control of self-organizing systems.” PhD Dissertation, Vrije Universiteit Brussel, Brussel, Belgium, 2007.

[10] C. Auer, P. Wüchner, and H. Meer, “A method to derive local interaction strategies for improving cooperation in self-organizing systems,” in *IWSOS ’08: Proceedings of the 3rd International Workshop on Self-Organizing Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 170–181.

[11] A. L. Nelson, E. Grant, and T. C. Henderson, “Evolution of neural controllers for competitive game playing with teams of mobile robots,” *Robotics and Autonomous Systems*, vol. 46, no. 3, pp. 135 – 150, 2004.

[12] G. Baldassarre, D. Parisi, and S. Nolfi, “Distributed coordination of simulated robots based on self-organization,” *Artif. Life*, vol. 12, no. 3, pp. 289–311, 2006.